Preprint
UCRL-JC-142207

Static Load Balancing for CFD Distributed Simulations

A. T. Chronopoulos, D. Grosu, A. M. Wissink, M. Benche

This article was submitted to
15th International Conference on Supercomputing, Sorreno, Italy,
June 18-21-, 2001

January 26, 2001

This work was performed under the auspices of the United States Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

Static Load Balancing for CFD Distributed Simulations

Anthony T. Chronopoulos*
Daniel Grosu†
Andrew M. Wissink‡
Manuel Benche§

Abstract

The cost/performance ratio of networks of workstations has been constantly improving. This trend is expected to continue in the near future. The aggregate peak rate of such systems often matches or exceeds the peak rate offered by the fastest parallel computers. This has motivated research towards using a network of computers, interconnected via a fast network (cluster system) or a simple Local Area Network (LAN) (distributed system), for high performance concurrent computations. Some of the important research issues arise such as (i) Optimal problem partitioning and virtual interconnection topology mapping; (ii) Optimal execution scheduling and load balancing.

CFD codes have been efficiently implemented on homogeneous parallel systems in the past. In particular, the helicopter aerodynamics CFD code TURNS has been implemented with MPI on the IBM SP with parallel relaxation and Krylov iterative methods used in place of more traditional recursive algorithms to enhance performance. In this implementation the space domain is divided into equal subdomain which are mapped to the processors. We

*Associate Professor, Division of Computer Science, University of Texas at San Antonio, 6900 North Loop 1604 West, San Antonio, TX 78249 atc@cs.utsa.edu

†PhD Student, Division of Computer Science, University of Texas at San Antonio, 6900 North Loop 1604 West, San Antonio, TX 78249

‡Computer Scientist, Member AIAA, Center for Applied Scientific Computing, Lawrence Livermore National Lab, P.O. Box 808, L-661, Livermore, CA 94551

§MSc. Student, Division of Computer Science, University of Texas at San Antonio, 6900 North Loop 1604 West, San Antonio, TX 78249

consider the implementation of TURNS on a LAN of heterogeneous workstations. In order to deal with the problem of load balancing due to the different processor speeds we propose a suboptimal algorithm of dividing the space domain into unequal subdomains and assign them to the different computers. The algorithm can apply to other CFD applications. We used our algorithm to schedule TURNS on a network of workstations and obtained significantly better results.

1 Introduction

Distributed computation offers the potential for cheaper and high performance computations. Some companies are beginning to utilize parallel processing in the form of clusters of workstations, that are idle during off-hours, to attain supercomputer performance [12]. Portable parallel software exist for implementing codes on distributed computer environments (e.g. Parallel Virtual Machine (PVM) or Message Passing Interface (MPI)) [5]). Several researchers have studied the problems of scheduling and load balancing computations for concurrent execution in distributed environments. For example see [1], [2], [4], [8], [10], [15] and the references therein.

The benefits of this research are immense for research agencies (e.g. NASA) and the broader scientific community. Design companies and government agencies have clusters of fast microcomputers, which are underutilized. Thus design or testing engineers could run these simulation codes on these clusters essentially free of cost.

1.1 A Helicopter Aerodynamics CFD Simulation Code

Accurate numerical simulation of the aerodynamics and aeroacoustics of rotary-wing aircraft is a complex and challenging problem. Three-dimensional unsteady Euler/Navier-Stokes computational fluid dynamics (CFD) methods are widely used (see [11] the references therein), but their application to large problems is limited by the amount of computer time they require. Such an example of a CFD application, which we will focus on, is the

computation of a helicopter aerodynamics. Efficient utilization of parallel processing is one effective means of speeding up these calculations [14]. The baseline numerical method is the structured-grid Euler/Navier-Stokes solver TURNS (Transonic Unsteady Rotor Navier Stokes) [11] developed in conjunction with the U.S. Army Aeroflightdynamics Directorate at NASA Ames Research Center. It is used for calculating the flowfield of a helicopter rotor (without fuselage) in hover and forward flight conditions. The governing equations solved by the TURNS code are the three-dimensional unsteady compressible thin-layer Navier-Stokes equations, applied in conservative form in a generalized body-fitted curvilinear coordinate system. The implicit operator used in TURNS for time-stepping in both steady and unsteady calculations is the Lower-Upper symmetric Gauss-Seidel (LU-SGS) operator of Yoon and Jameson [16].

The Hybrid LU-SGS is a parallel modification to LU-SGS [14]. Once the computational space has been divided into subdomains, the original LU-SGS algorithm is applied simultaneously to each processor subdomain. Then, border data between the subdomains is communicated using the relaxation-type approach of DP-LUR [14]. The use of multiple relaxation sweeps is retained to enhance robustness of the original algorithm lost in the domain decomposition. On a single processor, hybrid LU-SGS is identical to the original LU-SGS algorithm. On many processors (in the limit as the number of processors approaches the number of gridpoints), the algorithm becomes identical to DP-LUR (see [14] and references therein). Like DP-LUR, hybrid LU-SGS can be implemented such that it is completely load balanced with only nearest-neighbor communication required between the subdomains. Hybrid LU-SGS can be implemented such that it is completely load balanced with only nearest-neighbor communication between the sub-domains. In tests with transonic and supersonic problems with up to 512 subdomains, the hybrid LU-SGS method converges with a single relaxation sweep but the convergence rate is less than that of original LU-SGS. With two relaxation sweeps, the convergence rate is essentially identical to original LU-SGS. Further details of the hybrid LU-SGS algorithm are given in [14].

Inexact Newton methods coupled with Krylov subspace iterative methods for nonsymmetric linear systems have also been used. Many authors examined the Krylov methods for these applications see Ajmani [1] and McHugh and Knoll [7] and references there-in. These results concluded that the Arnoldi-based methods are the more efficient than the Biorthogonal methods. Wissink implemented the GMRES method and OSOMin [13] . Storage is a major consideration for the solution of three-dimensional problems. The Krylov methods require more storage than the LU-SGS method [14]. The LU-SGS method is used as a preconditioner in the Krylov methods to speed up their convergence rate.

We now review the parallel implementation of TURNS for a parallel system with homogeneous processors [13]. The time steping is serial. The three-dimensional flowfield spatial domain is divided in the wraparound and spanwise directions to form a two-dimensional array of processor subdomains, as shown in Fig. 1. Each processor executes a version of the code simultaneously for the portion of the flowfield that it holds. Coordinates are assigned to the processors to determine global values of the data each holds. Border data is communicated between processors, and a single layer of ghost-cells stores this communicated data. The Message Passing Interface (MPI) software routes communication between the processor subdomains.

TURNS approximates the solution at each time step based on two alternatives: (a) the relaxation (DP-LUR or LU-SGS) methods described above, or (b) the Inexact Newton Krylov methods. There are essentially four main steps of the inexact Newton algorithm [13]; (1) explicit (flux) function evaluation to form the right-hand-side vector, (2) preconditioning using hybrid LU-SGS (explained above), (3) implicit solution by the Krylov subspace solver, and (4) explicit application of boundary conditions. The (Jacobian-free) matrix multiplications are based on function evaluations in (3). Local processor communication is required in (1)-(4). We also have global communications in the error computation at each timestep and in the dotproducts in the Krylov methods.

The parallel implementation of TURNS with hybrid LU-SGS and OSOMin/GMRES

was performed on the IBM SP. Each processor was assigned a grid subdomain with equal number of grid points [13]. To deal with the heterogeneity of the processors we now consider subdiving the space domain into subdomains with unequal number of grid points.

The problem is how to obtain and map these partitions on a virtual mesh of processors. Only 2 dimensions (J and K) of the 3-dimensional grid($J \times K \times L$) are partitioned. The J dimension is partitoned into equal subpartitions and the K dimension is partitioned according to the power of each row of processors. The power of a PE row is defined as the power of the slowest PE on that row. Thus, evry PE is expected to complete execution in time proportional to the ratio of its load over processing power. Our algorithm checks every possible mesh configuration and proposes an optimal configuration that minimizes the execution time. Also our algorithm takes into consideration the memory size of each machine in making the allocation decision

Using our load balancing algorithm we were able to obtain an improvement in the speedup between 40% and 68% for LUSGS and 43% and 86% for GMRES, compared with the equal allocation method.

The remainder of this paper is organized as follows. In Section 2 we present one scheduling algorithm for homogeneous systems and one load balancing algorithm for heterogeneous systems. In section 3 we present the implementation and we discuss experimental results. Section 4 summarizes our results.

2 Scheduling and Load Balancing Algorithms

2.1 Algorithm for homogeneous processors

Assumptions:

1. We parallelize only the problem space domain and leave the time dimension for serial execution.

2. Processors (PEs) of the parallel system are of the same design and speed.

3. We assume a 2-D logical PE mesh with $p = r \times c$ PEs (Figure 1).

4. A *load* is measured as a 3-D box of grid points in the space domain.

5. Our goal is to assign equal loads to different PEs.

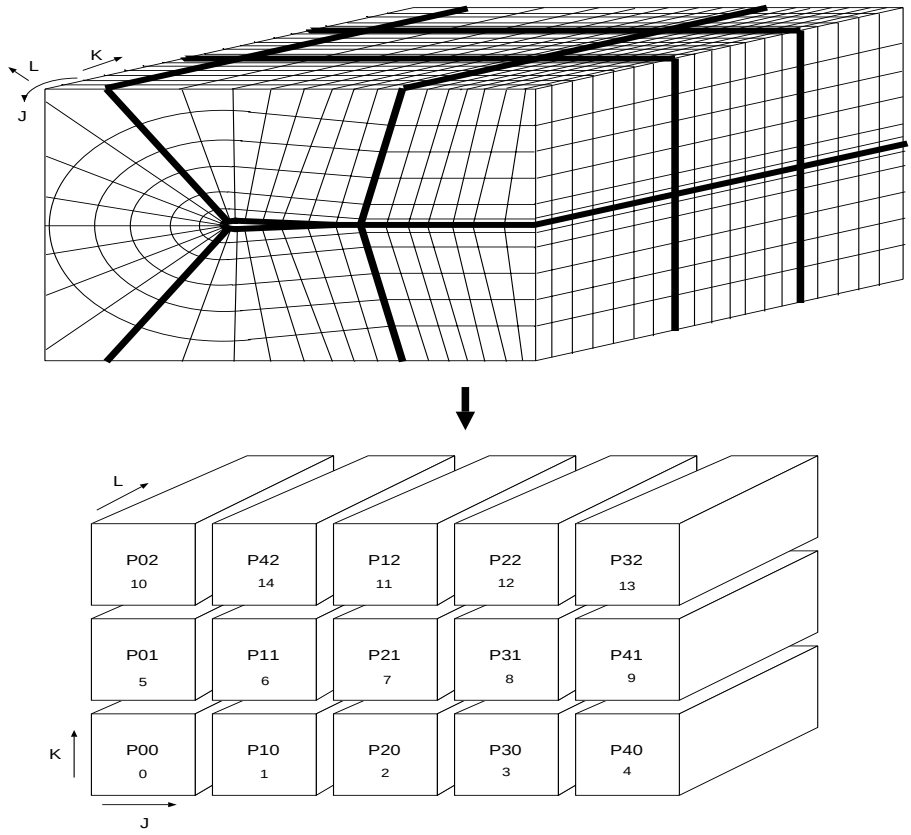

Figure 1: Partitioning the three-dimensional domain on a two-dimensional array of processors.

In mapping the domain of $J \times K \times L$ grid points to a logical 2-dimensional mesh of PEs the following restrictions apply:

(i) Because of the symmetric boundary condition applied at the airfoil surface in the J direction (data at $(j, *, 1)$ must equal data at $(J - j, *, 1)$), the same number of grid points are assigned to processors $P_{*,j}$ and $P_{*,J-j}$. For example this is only possible when J is odd.

(ii) The L dimension is not divided at all. We only partition the $J \times K$ mesh and assign $\frac{JK}{p}L$ grid points to each PE.

(iii) Each PE has only four adjacent PEs (implied by (ii)).

(iv) No PE can have fewer than 5 grid points assigned in each direction (J or K). The reason is that each PE has two shared boundary grid points with each of its four adjacent PEs (Figure 2).

In a mapping we may have four types of processor loads: $J_{load} \times K_{load} \times L$, $(J_{load}+1) \times K_{load} \times L$, $J_{load} \times (K_{load}+1) \times L$ and $(J_{load}+1) \times (K_{load}+1) \times L$ grid points (see Figure 3).

where,

$$J_{load} = \lfloor (J-2)/c + 2 \rfloor$$

$$K_{load} = \lfloor (K-2)/r + 2 \rfloor$$

$$L_{load} = L$$

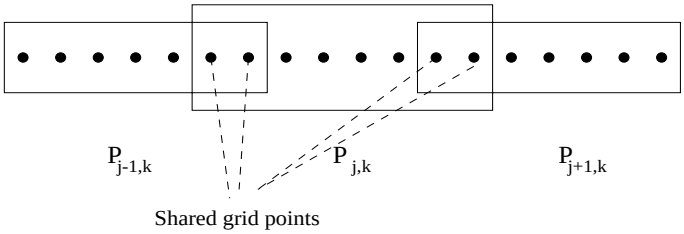

Figure 2: Shared boundary grid points.

We use as an estimate of the total execution time (T_{est}), the load corresponding to the largest value of these four loads. The goal is to minimize this value, by finding an optimal configuration $p = r \times c$ of processors. Figure 3 is an example with $p = r \times c = 3 \times 5$.

Remark:

Given a number of available PEs it is possible that a smaller number of PEs may produce a lower (T_{est}). For example, for $p = 59$, the best factorization ($r = 59$, $c = 1$) gives $J_{load} = 4$ and $K_{load} = 50$, and $T_{est} = (J_{load}+1) \times K_{load} = 250$. On the other hand, for $p = 58$, the best factorization (r = 29, c= 2) gives $J_{load} = 6$ and $K_{load} = 26$, thus $T_{est} = (J_{load}+1) \times K_{load} = 182$.

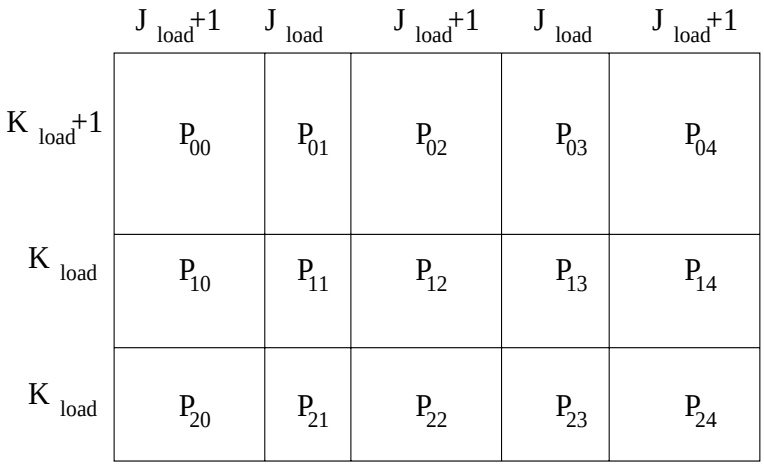

Figure 3: The loads for 3×5 processors configuration .

Given p PEs our algorithm checks every possible factorization of p and forms the logical PE mesh with the lowest T_{est}.

Algorithm 1:

for all (r, c) where $p = r \times c$ **do** $\min_{(r,c)} T_{est}(r, c)$

Function $T_{est}(r, c)$

$a = \lfloor (J-2)/c + 2) \rfloor$

$b = \lfloor (K-2)/r + 2) \rfloor$

$a_r = (J-2) \bmod c$

$b_r = (K-2) \bmod r$

if ((J is odd) **and** (c is even))

return error("symmetry restriction violated")

else

{Map the J direction to the c PEs of the k-th PE row }

if ($a_r \neq 0$ **and** a_r is odd)

map $J_{load} = (a+1)$ points to a_r PEs: $P_{k,0}, \ldots, P_{k,\frac{(a_r-1)}{2}}, P_{k,c/2}, P_{k,\frac{c-(a_r-1)}{2}}, \ldots, P_{k,c}$.

map $J_{load} = a$ points to remaining $(c - a_r)$ PEs

else a_r is even

map $J_{load} = (a+1)$ points to a_r PEs: $P_{k,0}, \ldots, P_{k,\frac{a_r}{2}}, P_{k,\frac{c-a_r}{2}}, \ldots, P_{k,c-1}$.

map $J_{load} = a$ points to remaining $(c - a_r)$ PEs

{Map the K direction to the r PEs of the j-th PE column}

map $K_{load} = (b+1)$ points to PEs: $P_{0,j}, \ldots, P_{b_r-1,j}$.

map $K_{load} = b$ points to remaining $(r - b_r)$ PEs.

{Compute T_{est}}

if $(a_r \neq 0)$

$T_{est} = a + 1$

else

$T_{est} = a$

if $(b_r \neq 0)$

$T_{est} = (b+1)T_{est}$

else

$T_{est} = b\, T_{est}$

return T_{est}

2.2 Algorithm for heterogeneous processors

Assumptions:

1. The assumptions of Algorithm 1.

2. We assume that the PEs of the parallel system are of different designs and speeds.

Given a number of processors $p = r \times c$, we divide the $J \times K$ sized grid in p rectangular partitions, with the J dimension divided in c subpartitions and the K dimension divided in r subpartitions. These subpartitions are possibly of unequal size.

3. We sort the PEs $(P_1, P_2, \ldots, P_p)$ in non-increasing order of their processing powers: $speed(P_0) \geq speed(P_1) \geq \ldots \geq speed(P_p)$ and assign the subpartition (j, k) to processor $P_{k \times J + j}$.

We expect the processors of a PE row to have approximately the same computing power. Thus, we divide the J dimension into equal subpartitions. The K direction is divided according to the power of each PE row, where the power of a PE row is defined as the power of the slowest PE on that row. This might result in slightly under-utilizing a more powerful processor, but avoids the more important issue of overloading a weak one.

Every PE is expected to complete execution in time proportional to the ratio of its load over processing power. We use as T_{est} the maximum of these ratios. The goal is to find a configuration (r, c) of PEs that minimizes T_{est}.

Our algorithm checks every possible factorization $p = r \times c$ of p PEs and proposes an optimal configuration with minimum T_{est} over all configurations under our assumptions. This is a suboptimal solution to the general grid partitioning problem see [3], [9] and references therein.

Note that it is possible that a configuration with a smaller number of processors can produce a smaller estimate value. The algorithm starts from the number q of available PEs and returns p PEs (where $p = r \times c$ and $p \leq q$) such that $(r, c) = \text{argmin } T_{est}$. Our heuristic approach checks these configurations by decrementing q by one at each stage.

Algorithm 2:

sort PEs $P_0 \ldots P_{p-1}$ in non-increasing order so that

$speed(P_0) \geq speed(P_1) \geq \ldots \geq speed(P_{p-1})$

for $p = q$ downto 1 **do**

for all (r, c) where $p = r \times c$ **do**

$\min_{(r,c)} T_{est}(r, c)$

return $p = r \times c = \text{argmin } T_{est}$

Function $T_{est}(r, c)$

$a = \lfloor (J - 2)/c + 2) \rfloor$

$a_r = (J-2) \bmod c$

if ((*J* is odd) **and** (*c* is even))

return error("symmetry violated")

else

{Obtain the minimum PE speed for each k-th PE row}

$row\_speed(k) = \min_{j=0\ldots c-1} speed(P_{k,j})$

{Compute total of all speeds}

$total\_speed = \sum_{k=0}^{r-1} row\_speed(k)$

{Map the J direction to the c PEs of the k-th PE row }

if ($a_r \neq 0$ **and** a_r is odd)

map $J_{load} = (a+1)$ points to a_r PEs: $P_{k,0}, \ldots, P_{k,\frac{(a_r-1)}{2}}, P_{k,c/2}, P_{k,\frac{c-(a_r-1)}{2}}, \ldots, P_{k,c}$.

map $J_{load} = a$ points to remaining $(c - a_r)$ PEs

else a_r is even

map $J_{load} = (a+1)$ points to a_r PEs: $P_{k,0}, \ldots, P_{k,\frac{a_r}{2}}, P_{k,\frac{c-a_r}{2}}, \ldots, P_{k,c-1}$.

map $J_{load} = a$ points to remaining $(c - a_r)$ PEs

{Map the K direction to the r PEs of the j-th PE column }

$b(k) = \frac{row\_speed(k)}{total\_speed} \times K$

$rem = K - \sum_{k=0}^{r-1} \lfloor b(k) \rfloor$

map $K_{load} = (\lfloor b(k) \rfloor + 1)$ points to rem PEs: $P_{0,j}, \ldots, P_{rem-1,j}$.

map $K_{load} = \lfloor b(k) \rfloor$ points to remaining $(r - rem)$ PEs.

{Compute T_{est}}

if ($a_r \neq 0$)

$J_{load} = a + 1$

else

$J_{load} = a$

if ($rem \neq 0$)

$K_{load} = \max_{k=0,\ldots,rem-1} (\lfloor b(k) \rfloor + 1)$

else

$$K_{load} = \max_{k=0,\ldots,r-1} (\lfloor b(k) \rfloor)$$

return $T_{est} = J_{load} \times K_{load}$

3 Implementation and results

3.1 Distributed Environment

In our experiments, we use a heterogeneous network of workstations which includes fourty eight SUN Ultra-10(440Mhz, 128MB), twelve SUN Ultra-1 (166Mhz, 64MB), one SGI-O2 (270Mhz, 128MB) and two SGI-O2 (200Mhz, 64MB). The SUN Ultra-10 and SGI workstations are connected to each other via a 100Mb/s switched Ethernet. All the other workstations are connected to each other via a 10Mb/s switched Ethernet. As a message-passing library we use the MPICH 1.2.0 [5]. For compiling the TURNS code on SUN workstations we use the Sun WorkShop Compiler FORTRAN 90 SPARC Version 2.0 and for SGI workstations we use the MIPS Pro FORTRAN 90 compiler.

3.2 Implementation details

We implemented the two algorithms described in previous section as two standalone programs written in C++. The first one considers the environment as a homogeneous network of workstations. The user has to provide the dimensons of the grid and the number of processors. The output is a parameter file used for compiling the TURNS code. The second program takes into consideration the heterogeneity of the computers. The user has to provide a list of machines with their speed and amount of memory in addition to the dimensions of the grid. The output is a parameter file used for compiling the TURNS code.

3.3 Experimental results

In order to analyse the performance of our algorithms we quantify the processing power of the heterogeneous distributed environment as a number of virtual processors (VP). One virtual processor is defined as the fastest processor in the system. In our case one virtual processor is equivalent to a SUN Ultra-10 (440Mhz, 128MB) workstation. For example if we have six SUN Ultra-10 (440Mhz, 128MB) workstations, one SGI-O2 (270Mhz, 128MB) and two SGI-O2 (200Mhz, 64MB) workstations the number of virtual processors is $VP = 7.5$.

We used the following notations:

- p - number of workstations;
- T_{comp} - computation time per integration step;
- T_{comm} - communication time per integration step;
- T_p - Execution time per integration step, $T_p = T_{comp} + T_{comm}$;

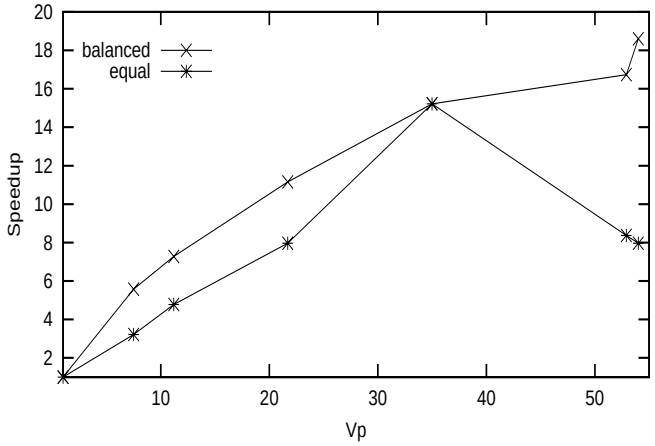

Figure 4: Speedup vs. number of virtual processors for LUSGS.

We run the code for both methods LUSGS and GMRES, and different configurations of workstations.

P (Vp)	Procs. mesh	Allocation method	LUSGS		GMRES	
			$T_{comm}(sec.)$	T_p(sec.)	$T_{comm}(sec.)$	T_p(sec.)
1 (1)	1x1	-	-	16.74	-	77.09
9 (7.5)	3x3	equal	3.2	5.2	15.6	25.3
		balanced	1.5	3.1	3.6	13.6
15 (11.2)	3x5	equal	2.5	3.5	9.4	15.3
		balanced	1.5	2.3	3.3	10.3
28 (21.7)	7x4	equal	1.5	2.1	5.6	8.9
		balanced	0.9	1.5	1.6	5.8
33 (23.6)	11x3	equal	1.7	2.1	5.9	8.6
		balanced	1.2	1.5	2.1	5.7
35 (35.0)	7x5	equal	0.7	1.1	2.06	4.3
		balanced	0.7	1.1	2.06	4.3
50 (41.0)	5x10	equal	1.6	2.0	5.4	7.5
		balanced	1.1	1.3	3.07	5.9
60 (52.9)	15x4	equal	1.2	2.0	5.8	7.1
		balanced	0.8	1.0	2.9	4.6
63 (54.0)	21x3	equal	1.25	2.1	5.4	6.6
		balanced	0.74	0.9	2.7	4.6

Table 1: Execution and communication time per integration step for LUSGS and GMRES using equal and load balanced allocation.

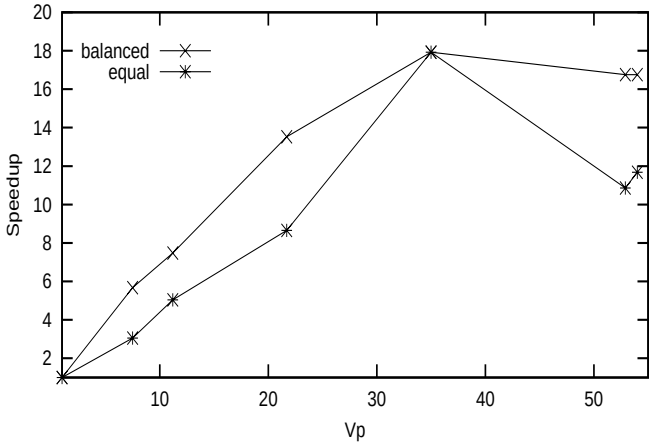

Figure 5: Speedup vs. number of virtual processors for GMRES.

The results of these runs are presented in Table 1. In order to compare the efficiency of our load balancer we used as a base line the execution time obtained using an equal allocation.

We compute the speedup according to the following equation[6]:

$$S_p = \frac{\min\{T_{P1}, T_{P2}, \ldots, T_{Pn}\}}{T_P}$$

where T_{Pi} is the execution time of the program on workstation Pi. The speedup for equal allocation and balanced allocation for the LUSGS method is presented in Figure 4.

Having a number of different workstations on our system the load balancer is able to determine the number of workstations for which we can obtain maximum performance. In our experiments the optimal configuration for a total of 63 workstations was determined by the load balancer to be formed by 60 workstation out of 63. Another example found by the load balancer is the configuration of 35 workstations out of 50. In those cases some of the slowest workstations were eliminated (by the load balancer) from the configuration (see Table 1) because their inclusion would lead to worse performance.

An important percentage of the execution time is due to the communication time. In Table 1 we shown the communication time for all configurations and allocation methods.

Configuration	Memory taken into account	T_p(sec.)	
		LUSGS	GMRES
$63 -> 60$	No	0.9	4.6
$63 -> 45$	Yes	0.9	3.7

Table 2: Execution time per integration step for LUSGS and GMRES using load balanced allocation.

Our load balancer takes into consideration the memory size of each machine in making the allocation decision. First the load balancer allocates the grid points according to Algorithm 2. We have added a technique in the load balancer which checks if the allocation exceeds the memory size of a machine. Such machines are eliminated from the distributed system and we get a lower q. Then a new allocation is computed using Algorithm 2. As an example we considered the q=63 processors case in which the load balancer without taking into account the memeory size suggests p=60 processors system as the best configuration. We run the load balancer considering the memory size and in this case it excludes 18 processors. By not utilizing 18 workstations, the number of workstations becomes p=45 and the execution time for GMRES is reduced from 4.6 seconds to 3.7 seconds. These results are shown in Table 2. As the table shows, the memory limitation appears only in the case of GMRES method.

4 Conclusion

In distributed simulations, the delivered performance of networks of heterogeneous computers degrades severely if the computations are not load balanced. In this work we consider the distributed simulation of a 3-D space CFD code (TURNS). We propose a load balancing heuristic for simulations on networks of heterogeneous computers. Our algorithm takes into account the CPU speed and memeory of the computers. Test run comparisons with the equal task allocation algorithm demonstrated significant efficiency gains.

ACKNOWLEDGEMENT: This research was supported, in part, by research grants from

(1) NASA NAG 2-1383 (1999-2000), (2) State of Texas Higher Education Coordinating Board through the Texas Advanced Research/Advanced Technology Program ATP 003658-0442-1999 (3) Air Force grant F49620-96-1-0472 (4) This research was also supported in part by NSF cooperative agreement ACI-9619020 through computing resources provided by the National Partnership for Advanced Computational Infrastructure at the University of California San Diego.

The third author was supported by a NASA Graduate Student Fellowship while the majority of this work was performed and is currently employed at the University of California Lawrence Livermore National Laboratory under U.S. Department of Energy contract number W-7405-Eng-48.

References

[1] K. Ajmani, M. S. Liou, and R. W. Dyson. Preconditioned Implicit Solvers for the Navier Stokes Equations on Distributed-Memory Machines. *AIAA paper 94-0408*, January 1994.

[2] C. A. Bohn and G. B. Lamont. Asymmetric Load Balancing on a Heterogeneous Cluster of PC. In *Proc. of the Intl. Conf. on Parallel and Distributed Processing Techniques and Applications*, pages 2512–2522, June 1999.

[3] P. E. Crandall and M. J. Quinn. Non-Uniform 2-D Grid Partitioning for Heterogeneous Parallel Architectures. In *Proc. of the 9th Intl. Parallel Processing Symp.*, pages 428–435, April 1995.

[4] T. Decker, R. Luling, and S. Tschoke. A Distributed Load Balancing Load Balancing Algorithm for Heterogeneous Parallel Computing Systems. In *Proc. of the Intl. Conf. on Parallel and Distributed Processing Techniques and Applications*, pages 933–940, June 1998.

[5] William D. Gropp and Ewing Lusk. *User's Guide for* `mpich`*, a Portable Implementation of MPI*. Mathematics and Computer Science Division, Argonne National Laboratory, 1996. ANL-96/6.

[6] D. Grosu. Some Performance Metrics for Heterogeneous Distributed Systems. In *Proc. of the Intl. Conf. on Parallel and Distributed Processing Techniques and Applications*, volume 5, pages 1261–1268, August 1996.

[7] P. R. McHugh and D. A. Knoll. Comparison of Standard and Matrix-Free Implementations of Several Newton-Krylov Solvers. *AIAA Journal*, 32(12):2394–2400, December 1994.

[8] A. Modi and L. N. Long. Unsteady separated Flow Simulations using a Cluster of Workstations. *AIAA Paper 2000-0272*, January 2000.

[9] D. M. Nicol. Rectilinear Partitioning of Irregular Data Parallel Computations. *J. of Parallel and Distributed Systems*, 23:119–134, 1994.

[10] Q. Snell, G. Judd, and M. Clement. Load Balancing in a Heterogeneous Supercomputing Environment. In *Proc. of the Intl. Conf. on Parallel and Distributed Processing Techniques and Applications*, pages 951–957, June 1998.

[11] G. R. Srinivasan and J.D. Baeder. TURNS: A Free-Wake Euler/Navier-Stokes Numerical Method for Helicopter Rotors. *AIAA Journal*, 31(5):959–962, May 1993.

[12] T. Sterling, T. Cwik, D. Becker, J. Salmon, M. Warren, and B. Nitzberg. An Assessment of Beowulf-Class Computing for NASA Requirements: Initial Findings from the First NASA Workshop on Beowulf-Class Clustered Computing. In *Proc. of the IEEE Aerospace Conference*, 1998.

[13] A. W. Wissink, A. S. Lyrintzis, and A. T. Chronopoulos. A Parallel Newton-Krylov Method for Rotary-wing Flowfield Calculations. *AIAA Journal*, 37(10):1213–1221, October 1999.

[14] A. W. Wissink, A. S. Lyrintzis, and R. C. Strawn. Parallelization of a Three-Dimensional Flow Solver for Euler Rotorcraft Aerodynamics Predictions. *AIAA Journal*, 34(11):2276–2283, November 1996.

[15] J. Xu and A. T. Chronopoulos. Distributed Self-Scheduling for Heterogeneous Workstation Clusters. In *Proc. of the ISCA 12th Int. Conf. on Parallel and Distributed Computing Systems*, pages 211–217, August 1999.

[16] S. Yoon and A. Jameson. A Lower-Upper Symmetric Gauss Seidel Method for the Euler and Navier Stokes Equations. *AIAA Journal*, 26:1025–1026, 1988.